Khare *et al. J Big Data (2017) 4:41*
DOI 10.1186/s40537-017-0102-7

Journal of Big Data

RESEARCH Open Access

HCudaBLAST: an implementation of BLAST on Hadoop and Cuda

Nilay Khare[1], Alind Khare[2] and Farhan Khan[1*]

*Correspondence:
farhan5900@gmail.com
[1] Maulana Azad National Institute of Technology, Bhopal, Madhya Pradesh 462003, India
Full list of author information is available at the end of the article

Abstract

The world of DNA sequencing has not only been a difficult field since it was first worked upon, but it is also growing at an exponential rate. The amount of data involved in DNA searching is huge, thereby normal tools or algorithms are not suitable to handle this degree of data processing. BLAST is a tool given by National Center for Biotechnology Information (NCBI) to compare nucleotide or protein sequences to sequence databases and calculate the statistical significance of matches. Many variants of BLAST such as blastn, blastp, blastx, etc. are used to search for nucleotides, proteins, nucleotides-to-proteins sequences respectively. GPU-BLAST and HBLAST have already been proposed to handle the vast amount of data involved in searching DNA sequencing and they also speedup the searching process. In this article, we propose a new model for searching DNA sequences—HCudaBLAST. It involves CUDA processing and Hadoop combined for efficient searching. The results recorded after implementing HCudaBLAST are shown. This solution combines the multi-core parallelism of GPGPUs and the scalability feature provided by the Hadoop framework.

Keywords: DNA Searching, BLAST, CUDA, Hadoop

Introduction

The DNA or protein sequence searching is the most obvious operation in the analysis of any new sequence and the reason for the same is pretty simple—finding similar regions of nucleotides or proteins between two or more nucleotide or protein sequences. The similarity can be used to determine many things including similarity of two or more species, identifying a completely new species, locating domains within the sequence of interest, etc. However, the difficulty in finding the similar regions between two or more sequences is very hard due to the size of the existing sequence involved. To overcome this difficulty, various tools or algorithms have been proposed. Let us look at some of these in the following paragraphs.

In computational biology and bioinformatics, aligning sequences to determine similarity between them is an essential and widely used computational procedure for biological sequences. There have been wide range of computational algorithms applied to the sequence alignment challenge. Methods like Smith–Waterman algorithm [1], which is quite slow but accurate and is based on dynamic programming, and, basic local alignment search tool (BLAST) [2] or FASTA [3] algorithm which is faster but less accurate and is based on heuristic or probabilistic programming. The very first algorithm was

given by Smith and Waterman in the form of Smith–Waterman algorithm in 1981. This is a global sequential alignment algorithm which involves high time complexity but at the same time, it gives optimal results. To overcome the time consumption of Smith–Waterman algorithm, Lipman and Pearson proposed FASTA tool in 1985, which takes a given nucleotide or amino acid sequence and searches a corresponding sequence database by using local sequence alignment. It is based on heuristic method which contributes to the high speed of its execution. It is also a heuristic algorithm like FASTA but it is more time-efficient as it searches only for the significant patterns in the sequences with comparative sensitivity.

Due to the heuristic approach, the execution speed of BLAST is significantly increased but the amount of data being processed is very large and is increasing at an exponential rate; such as UniMES for metagenomic data sets [4], which continues to expand exponentially as next generation sequencing (NGS). However, the solution for the scalability problem of the BLAST has been solved by combining Hadoop and BLAST together, which is called HBLAST [5]. It is a hybrid "virtual partitioning" approach that automatically adjusts the database partition size depending on the Hadoop cluster size as well as the number of input query sequences. Yet another improvement in the speed of BLAST algorithm has been proposed in the form of GPU-BLAST [6] or other CUDA based implementations [7]. GPU-BLAST can perform protein alignments up to four times faster than the single-threaded NCBI-BLAST. When GPU-BLAST is compared with six-threaded NCBI-BLAST then it performs nearly two times faster.

Hadoop is an open source framework developed by Apache and it is a platform that can store and manage large volumes of data and it also provides a relatively easy way to write and run applications for massive data processing. Hadoop basically works in two folds, the first fold is Hadoop distributed file system (HDFS) [8], which is a good fault-tolerant system for storing data using low-cost hardwares. The second fold is MapReduce [9], which is a programming model to process the stored data by running tasks with efficient scheduling. Hadoop works in the fashion of Master–Slave architecture and it comprises many daemon processes like NameNode, DataNode, NodeManager, ResourceManager, etcetera. HDFS comprises of a NameNode and numerous DataNodes, wherein the NameNode is in charge of administration of metadata, file blocks and namespace of HDFS-stored files and the DataNodes are in charge of physically storing and managing the data on the block level.

Hadoop provides scalability and gives efficient handling of existing database of sequences and GPU provides time efficiency for the whole process. It is intuitive to merge these two prominent technologies to gain the full strength of existing technologies and provide better results in terms of time and storage. So this article presents HCudaBLAST, which is a combination of Hadoop and CUDA processing that runs the existing NCBI BLAST algorithm.

The rest of the article is organized as follows. In "Background and related work" section, we introduce related work available in this topic. In "Proposed work" section, we give the points of interest of our proposed work and algorithm of HCudaBLAST, and in "Experimental setup" section, we give description of our experimental setup, followed by conclusion in "Conclusion" section.

Background and related work

BLAST with GPU computing

The graphics processing unit (GPU) is a multi-core processor that concurrently executes hundreds of threads. Originally, it was used for only graphics processing acceleration but recently, the GPUs are organized in a streaming, data parallel model in which it executes the same instructions on multiple data streams simultaneously. The basic architecture of GPU usually contains N GPU multiprocessors, each containing M processors. There are various frameworks accessible for GPU processing, yet the two most mainstream frameworks are compute unified device architecture (CUDA) and open computing language (OpenCL).

Multiple researches have been done to improve the time consumption of the BLAST algorithm by using GPU computing. Some of the most notable researches of the BLAST algorithm with GPUs are described below.

Panagiotis D. Vouzis et al

The above mentioned authors wrote an article "GPU-BLAST: using graphics processors to accelerate protein sequence alignment". They proposed an enhanced version of the popular NCBI-BLAST by using CUDA computing model. The GPU working involves the subject sequences to be stored in Global Memory whereas the queries are stored in Constant Memory. A GPU has multiple multiprocessors and each multiprocessor is represented by a block. These multiprocessors have multiple processors within them, which are represented by threads. Every multiprocessor has a shared memory. The working is such that the instruction that is to be performed is stored in the INSTRUCTION UNIT. The sequences are accessed from the global memory by processors and the score is calculated with the help of BLOSUM62 matrix, which is stored in the shared memory. The GPU-BLAST achieves speedup that is mostly between 3 and 4 in comparison to the sequential NCBI-BLAST, but it exhibits some drawbacks such as high power consumption with large amount of data.

BLAST with distributed computing

The size of the existing DNA or protein sequence database is so large that we cannot process it by a single computer system efficiently, no matter how much storage capacity and computing power the system has. Only distributed computing can provide a relatively better solution to deal with such kind of problems. There have been many researches in this area to explore the distributed computing for the BLAST algorithm. Most popular frameworks used in distributed computing are cloud computing and Hadoop framework. These frameworks have not only computing capability but also storage solutions for the large amount of data, so it can easily solve the scalability problem in the storage of existing database.

Andréa Matsunaga et al

The above mentioned authors wrote an article "CloudBLAST: combining MapReduce and virtualization on distributed resources for bioinformatics applications; [10]. They have implemented an integration of Hadoop, virtual workspaces, and virtual network (ViNe) as the MapReduce, virtual machine and virtual network technologies,

respectively, to deploy the commonly used bioinformatics tool NCBI BLAST. They have distributed query sequences in the cluster and processed them in a parallel fashion. First it creates a virtual machine (VM) and stores it in a virtual workspace factory server. When a biologist wants to process some query, he/she will have to request for the replica of the VM. The VM would be cloned and configured with the appropriate network addresses in each site. Now, the biologist can log into one the VMs, upload the input sequence and start executing BLAST job. The CloudBLAST does not work well with large clusters, we often need to do load balancing to gain the maximum performance.

Aisling O'Driscoll et al

The above mentioned authors wrote an article "HBLAST: parallelised sequence similarity—a Hadoop MapReducable basic local alignment search tool". They have used virtual partitioning to parallelize BLAST algorithm by partitioning both databases and input query sequences. It changes the database partition size depending on the cluster size of Hadoop and the number of input query sequences. The size of the virtual partition is decided by the size of the available random access memory (RAM), to overcome the slow I/O transfers. It uses makeblastdb tool to create physical partitions of the existing database files and each file is considered as a stand-alone database. By combining these files, we can create virtual partitions and store them in Hadoop data node. The HBLAST MapReduce algorithm takes a mapping file as an input, which consists of the name of the query file and all the files which belong to the single virtual partition in every line. After taking this file as an input, it will create the map task and compute the result for each query in single virtual partition and pass this result to the reduce task with subject sequence id, query sequence id and calculated score. The reduce task will aggregate the score on the basis of query sequence id and subject sequence id, and write them in a final file. HBLAST provides 2.5 times faster execution as opposed to CloudBLAST in case of large number of sequences.

Proposed work

As discussed, there are many variants of BLAST algorithm such as blastn, blastp, blastx, tblastx, etcetera. Here, the focus is only on implementation of blastp, which is used for searching protein sequences in an existing protein database. The blastp is already implemented with GPU in the aforementioned article and it reduces the searching time complexity. It is also already implemented with MapReduce in HBLAST and it also reduces the time consumption as well as it gives scalability to the storage of database. Here, presented an integration of these two solutions into one—an implementation of blastp algorithm with Hadoop and CUDA to take advantage of both the solutions.

The algorithm for HCudaBLAST is designed in such a way that it can efficiently merge these two parallel computing techniques. There are many libraries and tools available which provide the way to merge Hadoop with GPU. Each tool works in a different way and they have their own pros and cons. Following are the existing ways:

– JCUDA [11]: it provides Java bindings for Nvidia and related libraries, this will only work for GPUs by Nvidia.

– Java Aparapi [12]: it converts Java byte code to OpenCL at runtime. It also works with Hadoop and it supports different kind of GPUs.
– Native Code: it uses Java Native Interface (JNI) to call GPU code within Java code.

In these three methods mentioned above, Java Aparapi seems very prominent to use for Hadoop integration with GPUs, but it has many limitations and it also has to convert code from Java to OpenCL, so it will take more time. The Native Code which uses JNI, is relatively faster and has no limitations in the form of programming. It can be developed for both CUDA and OpenCL, that is why in this article Native Code method is used to implement HCudaBLAST algorithm for CUDA computing.

The working of HCudaBLAST is exceptionally straightforward. There is a mapping file as an input which consists of name of the query file and the name of the partition files of the database in each line with comma separation. This mapping file will be given to running job and it will be divided according to the number of mappers available. Each such mapper takes the part of the file as an input and processes each line at a time. In each iteration of the mapper, it will process the query file with given database partition files and generate all the seeds for the query with the word length of 3. Then the mapper will pass it to the process running on the GPU to calculate final alignments and collect the result generated by GPU process and then pass the query Id as a key & score with subject Id as a value to the reducer process.

The GPU process will basically take all the subject sequences, the query sequence, the generated seeds and the score matching matrix Blosum62 as inputs. It will then give each subject to a different thread and each thread will align the query sequence with subject sequence for each seed available. It will also extend the alignment of the seed till the score cut-off and expected cut-off are not met. This pair of the seed of query sequence and the part of the subject sequence with their score is called high scoring segment pairs (HSPs). By distributing tasks on all the processor available on GPU, it will parallelize the calculation of the HSPs and after calculating all the HSPs, it will return these HSPs to the mapper. After running the mapper process, Hadoop will do sorting and shuffling on the output generated by the mapper and all results will be merged according to the query Id, so the reducer will get the query Id as a key and lists of subject Ids and their scores as values. The reducer will collect all the score belonging to the same subject & the same query Id and calculate the sum of all those scores. It then gives query Id and subject Id as a key and their sum of score as a value, which is finally stored in an output file.

The tool used to partition the database is "makeblastdb". The benefit of partitioning the database is that it will properly distribute on a cluster and it will easily fit in an available memory. The block size available on Hadoop is 128 MB, that is why the maximum partition size of the database should be 128 MB, so that the time consumed by disk I/O could be reduced. The searching will be done for all the partitions, which are on a single node in a single mapper iteration. The formula to calculate expected score for limiting the alignment is $\text{eScore} = K * m * n * \mathbf{e}^{-s*\lambda}$, where n is the length of the query sequence, m is the length of the subject sequence, K and λ are statistical parameters estimated by fitting the distribution of local alignment scores and s is the current score. The algorithm for HCudaBLAST is appeared in Fig. 1.

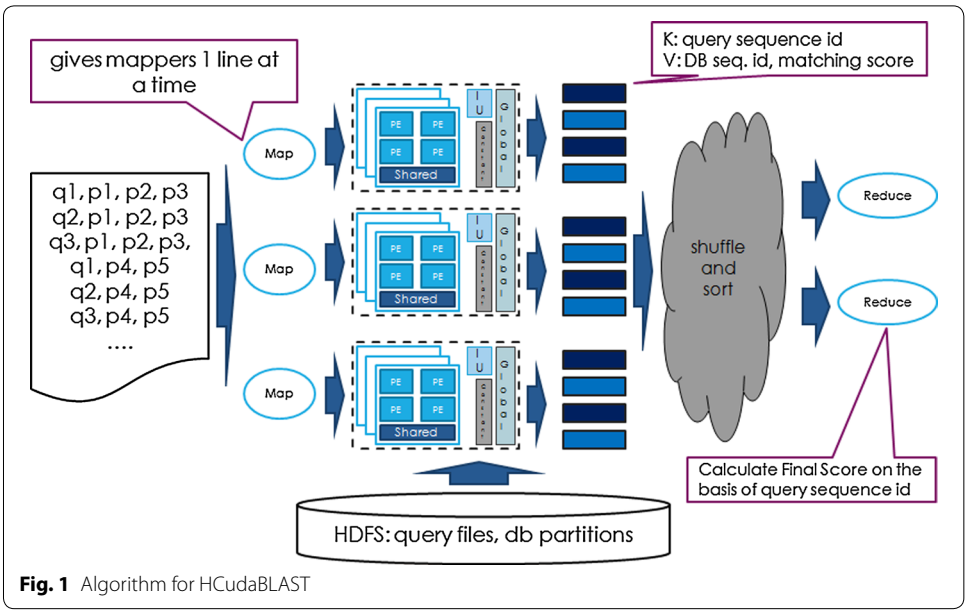

Fig. 1 Algorithm for HCudaBLAST

Experimental setup

The evaluation of the HCudaBLAST algorithm is done on a cluster of machines, setup with Hadoop framework and each such machine has Nvidia CUDA capability in it. Two machines in this cluster have Intel Xeon E5-2630 processor and 6 cores with 8 GB RAM and these machines have two CUDA devices, first one is Tesla C2075 with 14 multiprocessors and computing capability of 2.0 and second one is Quadro 2000 with four multiprocessor and computing capability of 2.1. Another three machines have Intel Xeon E31245 processor and 4 cores with 8 GB RARM and these machines have one CUDA device Quadro 600 with computing capability of 2.1. All the physical machines in a cluster was connected with a single switch in a local LAN. The cluster created here running Hadoop version 2.7 and CUDA toolkit 7.5 on each machine. Here, one machine act as a master node which runs NameNode and ResourceManager on it and all other four machines run DataNode and NodeManager on it. All the code run on Hadoop is written in Java programming language and the native code running on GPUs written in C, which called by Java using Native Interface (JNI).

The database comprises of approximate six million sequences of proteins and it has been taken from the NCBI web site [13]. The different size of query sequences have also been taken from same web site to perform the searches. This experiment has been performed in twofolds—one with two million sequences and the other with 6 million sequences with different size of queries and different number of partitions of the database to properly analyze the time taken by the algorithm.

Tables 1 and 2 show the results given by this experiment and the time taken by both Hadoop implementation of BLAST and HCudaBLAST. Obviously, HCudaBLAST shows a significant improvement in terms of time taken with different number of subject sequences and different size of query sequence.

Table 1 Performance of HCudaBLAST compared to generic BLAST algorithm on Hadoop with 2M sequences (BoH: BLAST on Hadoop, HCB: HCudaBLAST)

Query length	Database hits	BoH runtime (min)	HCB runtime (min)	Speed up
1000	148256049	12.43	7.33	1.69
2000	255375521	14.39	8.89	1.61
3000	593191597	15.69	9.99	1.57
4000	671434422	17.88	12.64	1.42
5000	829210234	19.32	14.88	1.30

Table 2 Performance of HCudaBLAST compared to generic BLAST algorithm on Hadoop with 6M sequences (BoH: BLAST on Hadoop, HCB: HCudaBLAST)

Query length	Database hits	BoH runtime (min)	HCB runtime (min)	Speed up
1000	2943252019	16.82	10.25	1.64
2000	3560375127	17.88	12.57	1.42
3000	4298763021	19.89	14.22	1.39
4000	5878829120	22.23	18.24	1.21
5000	7112991378	25.55	20.78	1.23

Conclusion

After seeing Tables 1 and 2, we can conclude the following:

- BLAST implementation on Hadoop and CUDA can indeed be combined together.
- The experimental setup that we had, consisted of differently configured machines in our cluster. It needs to be seen whether machines with same configuration in the cluster can give even better result then the one we got.
- In our experimental setup, we have machines with multiple cores on CPU itself, so we can make part of algorithm to use CPU multi-threading environment and gain better results.
- HCudaBLAST performs almost 1.5 times better than the individual Generic BLAST implementation on Hadoop.

Authors' contributions
Dr. NK carried out the molecular genetic studies, participated in the sequence alignment and drafted the manuscript. FK participated in the sequence alignment, the design of the study and performed the statistical analysis. AK conceived of the study, and participated in its design and coordination and helped to draft the manuscript. All authors read and approved the final manuscript.

Author details
[1] Maulana Azad National Institute of Technology, Bhopal, Madhya Pradesh 462003, India. [2] IIIT, Delhi, India.

Acknowledgements
We thank our colleagues from Maulana Azad National Institute of Technology, Bhopal who provided insight and expertise that greatly assisted the research. We also thank Dr. Akhtar Rasool, Assistant Professor; Mr. Amit Swami, Ph.D. Scholar; Mr. Gourav Hajela, Ph.D. Scholar and many more for their so-called insights in the form of reviews and immensely grateful comments on an earlier version of the manuscript, although any errors are our own and should not tarnish the reputations of these esteemed persons.

Competing interests
The authors declare that they have no competing interests.

Availability of data and materials
The data used in this article is publicly available in NCBI repository of protein databases and the link for the same is mentioned in the references section.

Consent for publication
Not applicable.

Ethics approval and consent to participate
Not applicable.

Funding
We are presenting this article without any kind of funding. We will be more than happy if you provide us some discount in the matter and give us adequate time period for making the arrangement of fee.

Publisher's Note

Received: 8 December 2016 Accepted: 7 November 2017
Published online: 16 November 2017

References

1. Smith TF, Waterman MS. Identification of common molecular subsequences. J Mol Biol. 1981;147(1):195–7.
2. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. Basic local alignment search tool. J Mol Biol. 1990;215(3):403–10.
3. Lipman DJ, Pearson WR. Rapid and sensitive protein similarity searches. Science. 1985;227(4693):1435–41.
4. Sleator RD, Shortall C, Hill C. Metagenomics. Lett Appl Microbiol. 2008;47(5):361–6.
5. O'Driscoll A, Belogrudov V, Carroll J, Kropp K, Walsh Paul, Ghazal Peter, Sleator Roy D. HBLAST: parallelised sequence similarity—a hadoop MapReducable basic local alignment search tool. J Biomed Inform. 2015;54:58–64.
6. Vouzis PD, Sahinidis NV. GPU-BLAST: using graphics processors to accelerate protein sequence alignment. Bioinformatics. 2011;27(2):182–8.
7. Manavski SA, Valle G. CUDA compatible GPU cards as efficient hardware accelerators for Smith–Waterman sequence alignment. BMC Bioinform. 2008;9(2):1.
8. Shafer J, Rixner S, Cox AL. The hadoop distributed filesystem: balancing portability and performance. In: IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS); 2010. p. 122–33.
9. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Commun ACM. 2008;51(1):107–13.
10. Matsunaga A, Tsugawa M, Fortes J. Cloudblast: combining mapreduce and virtualization on distributed resources for bioinformatics applications. In: IEEE Fourth International Conference on eScience; 2008. p. 222–9.
11. Yan Y, Grossman M, Sarkar V. JCUDA: a programmer-friendly interface for accelerating Java programs with CUDA. In: Euro-Par 2009 Parallel Processing; 2009. p. 887–99.
12. Grossman M, Breternitz M, Sarkar V. Hadoopcl: Mapreduce on distributed heterogeneous platforms through seamless integration of hadoop and opencl. In: IEEE 27th International on Parallel and Distributed Processing Symposium Workshops & Ph.D. Forum (IPDPSW); 2013. p. 1918–27.
13. NCBI Database. ftp://ftp.ncbi.nih.gov/blast/db/. Accessed 29 Nov 2016.